On the Power of Multidoubling in Speeding Up Elliptic Scalar Multiplication

Yasuyuki Sakai[1] and Kouichi Sakurai[2]

[1] Mitsubishi Electric Corporation,
5-1-1 Ofuna, Kamakura, Kanagawa 247-8501, Japan
ysakai@iss.isl.melco.co.jp
[2] Kyushu University,
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
sakurai@csce.kyushu-u.ac.jp

Abstract. We discuss multidoubling methods for efficient elliptic scalar multiplication. The methods allows computation of $2^k P$ directly from P without computing the intermediate points, where P denotes a randomly selected point on an elliptic curve. We introduce algorithms for elliptic curves with Montgomery form and Weierstrass form defined over finite fields with characteristic greater than 3 in terms of affine coordinates. These algorithms are faster than k repeated doublings. Moreover, we apply the algorithms to scalar multiplication on elliptic curves and analyze computational complexity. As a result of our implementation with respect to the Montgomery and Weierstrass forms in terms of affine coordinates, we achieved running time reduced by 28% and 31%, respectively, in the scalar multiplication of an elliptic curve of size 160-bit over finite fields with characteristic greater than 3.

Keywords. Elliptic curve cryptosystems, Scalar multiplication, Montgomery form, Multidoubling, Fast implementation

1 Introduction

Elliptic curve cryptosystems, which were suggested by Miller [Mi85] and Koblitz [Ko87], are now widely used in various security services. IEEE and other standardizing bodies such as ANSI and ISO are in the process of standardizing elliptic curve cryptosystems. Therefore, it is very attractive to provide algorithms that allow efficient implementation. Encryption/decryption or signature generation/verification schemes require computation of scalar multiplication. The computational performance of cryptographic protocols with elliptic curves strongly depends on the efficiency of the scalar multiplication. Thus, fast scalar multiplication is essential for elliptic curve cryptosystems.

One method to increase doubling speed involves the "*multidoubling*", which computes $2^k P$ directly from $P \in E(\mathbb{F}_q)$, without computing the intermediate points $2P,\ 2^2 P, \cdots, 2^{k-1} P$. The concept of multidoubling was first suggested by Guajardo and Paar in [GP97]. They formulated algorithms for the multidoubling

S. Vaudenay and A. Youssef (Eds.): SAC 2001, LNCS 2259, pp. 268–283, 2001.

of $4P$, $8P$ and $16P$ on elliptic curves over $\mathbb{F}_{2^n}$ in terms of affine coordinates. Recent related results include a formula for computing $4P$ on elliptic curves over $\mathbb{F}_p$ in affine coordinates by Müller [Mu97] and a formula for computing $4P$ on elliptic curves over $\mathbb{F}_p$ in projective coordinates by Miyaji, Ono and Cohen [MOC97a]. These formulae are more efficient than repeated doublings. All of the previous works were on the subject of elliptic curves with Weierstrass form. Another model of an elliptic curve that is useful for cryptosystems is the Montgomery form. Montgomery introduced the equation to speed up integer factorization with elliptic curves [Mo87]. The elliptic curve method of factoring was proposed by H.W.Lenstra [Le87]. In recent years, several authors have proposed elliptic curve cryptosystems using the Montgomery model [Iz99,LD99,OKS00].

In this paper, we propose efficient algorithms for speeding up elliptic curve cryptosystems with Montgomery elliptic curves in terms of affine coordinates. We construct efficient formulae that compute 2^kP directly for $\forall k \geq 2$. In the case of an elliptic curve with Montgomery form, our formulae have computational complexity $(8k+4)\mathcal{M}+(4k-1)\mathcal{S}+\mathcal{I}$, where $\mathcal{M}$, $\mathcal{S}$, and $\mathcal{I}$ denote multiplication, squaring and inversion in $\mathbb{F}_p$, respectively. This is more efficient than k repeated doublings, which require k inversions. When implementing our multidoubling method, experimental results show that computing $16P$ achieved running time reduced by 40% over 4 doublings in affine coordinates. Moreover we introduce formulae that compute 2^kP directly for $\forall k \geq 2$, for Weierstrass elliptic curves in terms of affine coordinates. Our formulae have computational complexity $(4k+1)\mathcal{M}+(4k+1)\mathcal{S}+\mathcal{I}$. The formulae have slightly simple form compared to the formulae described in [SS01] and have computational advantage, due to one field multiplication, over the formulae proposed in [SS00]

As a results of our implementation with respect to Montgomery and Weierstrass forms in terms of affine coordinates, we achieved running time reduced by 28% and 31%, respectively, in the scalar multiplication of an elliptic curve of size 160-bit. We also discuss the computational complexity of scalar multiplication using multidoubling. The proposed algorithm improve the performance of scalar multiplication with the binary method, as well as the window method. Therefore, they are effective in restricted environments where resources are limited, such as smart cards.

2 Previous Work

In this section, we summarize the multidoubling, the direct computation and arithmetic for an elliptic curve with Montgomery form.

2.1 Multidoubling and Direct Computation

The concept of using multidoubling and direct computation of 2^kP to efficiently implement elliptic scalar multiplication was first proposed by Guajardo and Paar in [GP97]. They formulated algorithms for computing $4P$, $8P$, and $16P$ on elliptic curves over $\mathbb{F}_{2^n}$ in terms of affine coordinates. In recent years, several authors

have reported methods that compute $2^k P$ directly, but some of them are limited to small k. The following section summarizes the previous work on multidoubling and direct computation.

1. Guajardo and Paar [GP97] proposed formulae for computing $4P$, $8P$, and $16P$ on elliptic curves over $\mathbb{F}_{2^n}$ in terms of affine coordinates.
2. Müller [Mu97] proposed formulae for computing $4P$ on elliptic curves over $\mathbb{F}_p$ in terms of affine coordinates.
3. Miyaji, Ono, and Cohen [MOC97a] proposed formulae for computing $4P$ on elliptic curves over $\mathbb{F}_p$ in terms of projective coordinates.
4. Han and Tan [HT99] proposed formulae for computing $3P$, $5P$, $6P$, $7P$, etc, on elliptic curves over $\mathbb{F}_{2^n}$ in terms of affine coordinates.
5. Sakai and Sakurai [SS00,SS01] proposed formulae for computing $2^k P$ ($\forall k \geq 1$) on elliptic curves over $\mathbb{F}_p$ in terms of affine coordinates.

We should remark that the algorithm proposed by Cohen, Miyaji and Ono in [CMO98] can be efficiently used for the direct computation of several doublings. The authors call their algorithm a "*modified jacobian*" coordinate system. The coordinate system uses (redundant) mixed representation such as (X, Y, Z, aZ^4). Doubling in terms of the modified jacobian coordinates has computational advantages over weighted projective (jacobian) coordinates. Itoh et al. also gave a similar method for doubling [ITTTK99].

All of these works dealt with computations on Weierstrass elliptic curves. In later sections, we will formulate algorithms that work on Montgomery elliptic curves in terms of affine coordinates, and analyze their computational complexity.

2.2 Elliptic Curves with Montgomery Model

Let $a, b \in \mathbb{F}_p$, $4a^3 + 27b^2 \neq 0$, $p > 3$, and p be a prime number. An elliptic curve defined over $\mathbb{F}_p$ for Weierstrass model is defined by the following equation (1). Elliptic curve cryptosystems using curves with Weierstrass form are in the process of being standardized, e.g., [IEEE], and are widely used in various security services.

$$E : y^2 = x^3 + ax + b \tag{1}$$

H. W. Lenstra proposed the elliptic curve method of factoring [Le87]. Montgomery introduced the following equation to speed up integer factorization with elliptic curves [Mo87]. In recent years, several authors have proposed cryptosystems using an elliptic curve with Montgomery form [Iz99,LD99,OKS00]. Let $A, B \in \mathbb{F}_p$, $(A^2 - 4)B \neq 0$. An elliptic curve of Montgomery model is defined by the following equation (2).

$$E_m : Bv^2 = u^3 + Au^2 + u \tag{2}$$

The formulae for transforming Montgomery and Weierstrass forms are given by the following (See [Iz99] for details).

By the transformation $u = \frac{3x-AB}{3B}$ and $v = \frac{y}{B^2}$, we obtain $y^2 = x^3 + \frac{B^2(3-A^2)}{3}x + \frac{AB^3(2A^2-9)}{27}$. Therefore, by the relationship $a = \frac{B^2(3-A^2)}{3}$ and $b = \frac{AB^3(2A^2-9)}{27}$, we can transform a Montgomery form into a Weierstrass form.

The above linear transformation clearly converts any elliptic curve with Montgomery form into a curve with Weierstrass form. However, the inverse transformation, from Weierstrass form to Montgomery form, works only if there exists a particular curve. Based on the above relationship between (a, b) and (A, B), we eliminate B, then we obtain $A^6 - 9A^4 - 27(r-1)A^2 + 27(4r-1) = 0$, where $r = \frac{a^3}{4a^4+27b^2}$. Let we consider the equation $f(t) = t^3 - 9t^2 - 27(r-1)t + 27(4r-1) = 0$, where $t = A^2$. If $f(t)$ has a solution $t = \alpha$ such as a quadratic residue in $\mathbb{F}_p$, a Weierstrass form can be transform into a Montgomery form by the following relation. Let β be a square root of α, we obtain $A = \beta$ and $B = \frac{9b(3-\beta^2)}{\alpha\beta(2\beta^2-9)}$. Then the relation $x = \frac{3Bu+AB}{3}$ and $y = B^2v$ are derived.

A detailed analysis on the case which can transform a Weierstrass form into a Montgomery form was given by Izu [Iz99]. He concluded that approximately 40% of curves with Weierstrass form can be transformed into a curve with Montgomery form.

2.3 Group Operation for Elliptic Curves with Montgomery Form

We describe algorithms for group operation in an elliptic curve with Montgomery form. When we estimate a computational efficiency, we will ignore the cost of a field addition, as well as the cost of a multiplication by small constants.

Affine Coordinates. Suppose $P_3(u_3, v_3) = P_1(u_1, v_1) + P_2(u_2, v_2) \in E_m(\mathbb{F}_p)$, and $P_1 \neq P_2$. The addition formulae are given by the following.

$$\begin{aligned} u_3 &= B\lambda^2 - A - u_1 - u_2 \\ v_3 &= \lambda(u_1 - u_3) - v_1 \\ \lambda &= \frac{v_1 - v_2}{u_1 - u_2} \end{aligned} \tag{3}$$

The computational complexity for an addition involves $3\mathcal{M} + \mathcal{S} + \mathcal{I}$.

Suppose $P_3(u_3, v_3) = 2P_1(u_1, v_1) \in E_m(\mathbb{F}_p)$. Point doubling can be accomplished by the following.

$$\begin{aligned} u_3 &= B\lambda^2 - A - 2u_1 \\ v_3 &= \lambda(u_1 - u_3) - v_1 \\ \lambda &= \frac{3u_1^2 + 2Au_1 + 1}{2Bv_1} \end{aligned} \tag{4}$$

The computational complexity of a doubling involves $5\mathcal{M} + 2\mathcal{S} + \mathcal{I}$.

Projective Coordinates. Next, we describe formulae for the group operations in projective coordinates. Let $u = \frac{U}{W}$, $v = \frac{V}{W}$. Suppose $P_2(U_2, W_2) =$

$P_1(U_1, W_1) + P(u, v, 1)$. The point $P_3(U_3, W_3) = P_1(U_1, W_1) + P_2(U_2, W_2)$ can be computed by the following.

$$\begin{aligned} U_3 &= (U_1U_2 - W_1W_2)^2 \\ &= (Sub_1Add_2 + Add_1Sub_2)^2 \\ W_3 &= u\,(U_1W_2 - W_1U_2)^2 \\ &= u\,(Sub_1Add_2 - Add_1Sub_2)^2 \end{aligned}$$

where, $Add_1 = U_1 + W_1$, $Sub_1 = U_1 - W_1$, $Add_2 = U_2 + W_2$ and $Sub_2 = U_2 - W_2$. The computational complexity for an addition involves $3\mathcal{M} + 2\mathcal{S}$.

Note that V-coordinate does not enter into any of the formulae. An addition can be accomplished without computation of the V-coordinate if the difference between the two given points is known [Mo87]. Point doubling can be accomplished by the following.

$$\begin{aligned} U_3 &= \left(U_1^2 - W_1^2\right)^2 = Add_1^2 Sub_1^2 \\ W_3 &= 4U_1W_1\left(U_1^2 + AU_1W_1 + W_1^2\right) \\ &= \left(Add_1^2 - Sub_1^2\right)\left\{Sub_1^2 + C\left(Add_1^2 - Sub_1^2\right)\right\} \end{aligned}$$

where, $C = \frac{A+2}{4}$. For a given curve, C can be pre-computed. Therefore above formulae have computational complexity $3\mathcal{M} + 2\mathcal{S}$.

The basic well known method for elliptic scalar multiplication on curves with Weierstrass form is the "*double-and-add*" (or binary) method. There are several methods which have computational advantage over the binary method such as the window method. However, in the case of elliptic curves with Montgomery form in terms of projective coordinates, we can not apply such methods, because the difference between the two given points, i.e., the U-coordinate of $P_2 - P_1$, must be known when adding the two points. To compute kP, we compute $2P$ and then repeatedly compute two points $(2mP, (2m+1)P)$ or $((2m+1)P, (2m+2)P)$, depending on whether the corresponding bit in the binary representation of k is a 0 or a 1 [AMV93,Mo87,MV93]. This method maintains the invariant relationship such that the difference of the two points always P.

3 The Proposed Algorithms

In this section, we describe new algorithms for elliptic curves with Montgomery form, which compute 2^kP directly from a given point $P \in E_m(\mathbb{F}_p)$ without computing the intermediate points $2P,\ 2^2P, \cdots, 2^{k-1}P$. We will begin by constructing formulae for small k, then we will construct an algorithm for general k ($k \geq 2$). We will also show an algorithm that compute 2^kP directly for elliptic curves with Weierstrass form. This is an improved version of the algorithm proposed in [SS00,SS01].

3.1 Montgomery Form

As an example, we give an algorithm that compute $8P$ directly from $P \in E_m(\mathbb{F}_p)$.

Computing 8*P*. Let $P_8(u_8, v_8) = 8P(u_1, v_1) \in E_m(\mathbb{F}_p)$. For an elliptic curve with Montgomery form in terms of affine coordinates, P_8 can be computed in the following way. The derivation is based on repeated substitution of the point doubling formulae, such that only one field inversion needs to be calculated. First we compute C, D_i, E_i, F_i, for $1 \leq i \leq 3$ as follows.

$$\begin{aligned}
C &= AB \\
D_1 &= u_1 B \\
E_1 &= v_1 \\
F_1 &= 1 + u_1(2A + 3u_1) \\
D_2 &= -2^2 C E_1^2 - 8 D_1 E_1^2 + F_1^2 \\
E_2 &= -8B^2 E_1^4 + F_1(4 D_1 E_1^2 - D_2) \\
F_2 &= 2^4 B^2 E_1^4 + D_2(2^3 C E_1^2 + 3 D_2) \\
D_3 &= -2^4 C (E_1 E_2)^2 - 8 D_2 E_2^2 + F_2^2 \\
E_3 &= -8 E_2^4 + F_2(4 D_2 E_2^2 - D_3) \\
F_3 &= 2^8 B^2 (E_1 E_2)^4 + D_3(2^5 C (E_1 E_2)^2 + 3 D_3)
\end{aligned}$$

Then we compute u_8 and v_8 as follows.

$$\begin{aligned}
u_8 &= \frac{-8E_3^2(2^3 C(E_1E_2)^2 + D_3) + F_3^2}{2^6 B (E_1 E_2 E_3)^2} \\
v_8 &= \frac{F_3((2^6 C(E_1E_2)^2 + 12 D_3)E_3^2 - F_3^2) - 8E_3^4}{2^9 B^2 (E_1 E_2 E_3)^3}
\end{aligned}$$

Note that C and B^2 can be pre-computed, and that although the denominator of u_8 differs from that of v_8, the above formulae require only one inversion if we multiply the numerator of u_8 by $2^3 B E_1 E_2 E_3$. The above formulae have computational complexity $25\mathcal{M} + 11\mathcal{S} + \mathcal{I}$.

Multidoubling. From the formulae that compute $2^k P$ for small k, given in the previous subsection, we can easily obtain general formulae that allow direct doubling $P \mapsto 2^k P$ for $k \geq 2$. The figure shown below describes the formulae, and their computational complexity is given as Theorem 1.

Algorithm 1: Direct computation of $2^k P$ in affine coordinates on an elliptic curve with Montgomery form, where $k \geq 2$ and $P \in E_m(\mathbb{F}_p)$.

`INPUT:` $P_1 = (u_1, v_1) \in E_m(\mathbb{F}_p)$
`OUTPUT:` $P_{2^k} = 2^k P_1 = (u_{2^k}, v_{2^k}) \in E_m(\mathbb{F}_p)$

`Pre Computations`

$$\begin{aligned}
C &= AB \\
B_2 &= B^2
\end{aligned}$$

`Step 1.` Compute D_0, E_0 and F_0

$$\begin{aligned}
D_0 &= u_1 B \\
E_0 &= v_1 \\
F_0 &= 1 + u_1(2A + 3u_1)
\end{aligned}$$

`Step 2.` For i from 1 to k compute D_i and E_i, for i from 1 to $k-1$ compute F_i

$$D_i = -2^{2i}C\left(\prod_{j=0}^{i-1} E_j\right)^2 - 8D_{i-1}E_{i-1}^2 + F_{i-1}^2$$

$$\texttt{if } i = 1 \quad E_1 = -8B_2E_0^4 + F_0(4D_0E_0^2 - D_1)$$

$$\texttt{else} \qquad E_i = -8E_{i-1}^4 + F_{i-1}(4D_{i-1}E_{i-1}^2 - D_i)$$

$$F_i = \left(\prod_{j=0}^{i-1} E_j\right)^2 \left(2^{4i}B_2\left(\prod_{j=0}^{i-1} E_j\right)^2 + 2^{2i+1}CD_i\right) + 3D_i^2$$

`Step 3.` Compute u_{2^k} and v_{2^k}

$$u_{2^k} = \frac{D_k}{B\left(2^k \prod_{i=0}^{k-1} E_i\right)^2}$$
$$v_{2^k} = \frac{E_k}{B_2\left(2^k \prod_{i=0}^{k-1} E_i\right)^3}$$

Theorem 1. *For an elliptic curve with Montgomery form in terms of affine coordinates, there exists an algorithm that computes 2^kP, with $k \geq 2$, in at most $8k+4$ field multiplication, $4k-1$ field squaring and one inversion in $\mathbb{F}_p$ for any point $P \in E_m(\mathbb{F}_p)$, excluding precomputation.*

The proof is outlined in Appendix A.1.

3.2 Weierstrass Form

The multidoubling for Weierstrass elliptic curves in terms of affine coordinates is given below. Their computational complexity have, given as Theorem 2, $(4k+1)\mathcal{M}+(4k+1)\mathcal{S}+\mathcal{I}$. The complexity has a one field multiplication computational advantage over the formulae proposed in [SS00]. Moreover, the formulae have slightly simple form compared to the formulae described in [SS01]

Algorithm 2: Direct computation of 2^kP in affine coordinates on an elliptic curve with Weierstrass form, where $k \geq 1$ and $P \in E(\mathbb{F}_p)$.

`INPUT:` $P_1 = (x_1, y_1) \in E(\mathbb{F}_p)$
`OUTPUT:` $P_{2^k} = 2^kP_1 = (x_{2^k}, y_{2^k}) \in E(\mathbb{F}_p)$

`Step 1.` Compute A_0, B_0 and C_0

$$A_0 = x_1$$
$$B_0 = y_1$$
$$C_0 = 3x_1^2 + a$$

`Step 2.` For i from 1 to k compute A_i and B_i, for i from 1 to $k-1$ compute C_i

$$\begin{aligned} A_i &= C_{i-1}^2 - 8A_{i-1}B_{i-1}^2 \\ B_i &= 8B_{i-1}^4 + C_{i-1}(A_i - 4A_{i-1}B_{i-1}^2) \\ C_i &= 3A_i^2 + 16^{i-1}a\left(\prod_{j=1}^{i-1} B_j\right)^4 \end{aligned}$$

`Step 3.` Compute x_{2^k} and y_{2^k}

$$\begin{aligned} x_{2^k} &= \frac{A_k}{\left(2^k \prod_{i=1}^{k} B_i\right)^2} \\ y_{2^k} &= \frac{B_k}{\left(2^k \prod_{i=1}^{k} B_i\right)^3} \end{aligned}$$

Theorem 2. *For an elliptic curve with Weierstrass form in terms of affine coordinates, there exists an algorithm that computes $2^k P$ in at most $4k+1$ multiplications, $4k+1$ squarings, and one inversion in $\mathbb{F}_p$ for any point $P \in E(\mathbb{F}_p)$.*

The proof is outlined in Appendix A.2.

3.3 Complexity Comparison on Direct Computation

In this subsection, we compare the computational complexity of the multidoubling given in the previous subsection with the complexity of k separate repeated doublings. The complexity of a doubling is estimated from the algorithm given by the formulae (4) or as shown in [IEEE]. Tables 1 and 2 show the number of multiplications $\mathcal{M}$, squarings $\mathcal{S}$, and inversions $\mathcal{I}$ in the base field $\mathbb{F}_p$. Note that our method reduces inversions at the cost of multiplications. Therefore, the performance of the new formulae depends on the cost factor of one inversion relative to one multiplication. For this purpose, we introduce the notation of a "*break-even point*", as used in [GP97]. It is possible to express the time that it takes to perform one inversion in terms of the equivalent number of multiplications needed per inversion. In this comparison, we assume that one squaring has complexity $\mathcal{S} = 0.8\mathcal{M}$, and that the costs of field addition and multiplication by small constants can be ignored.

As we can see from Table 1, if a field inversion has complexity $\mathcal{I} > 10.4\mathcal{M}$, one quadrupling will be more efficient than two separate doublings. If $\mathbb{F}_p$ has size 160-bit or larger, it is likely that $\mathcal{I} > 10.4\mathcal{M}$ in many implementations (e.g., see [WMPW98]). In addition, if $k > 2$, our direct computation method is more efficient than individual doublings in most implementations. For Weierstrass form, shown in Table 2, our direct computation method is more efficient than individual doublings in most implementations.

Table 1. Complexity comparison on direct computation : Montgomery form

Calculation	Method	Complexity			Break-Even Point
		$\mathcal{M}$	$\mathcal{S}$	$\mathcal{I}$	
$4P$	Direct computation	18	7	1	$10.4\mathcal{M} < \mathcal{I}$
	Separate 2 doublings	10	4	2	
$8P$	Direct computation	25	11	1	$7.0\mathcal{M} < \mathcal{I}$
	Separate 3 doublings	15	6	3	
$16P$	Direct computation	32	15	1	$5.9\mathcal{M} < \mathcal{I}$
	Separate 4 doublings	20	8	4	
2^kP	Direct computation	$8k+4$	$4k-1$	1	$(4.6+\frac{7.8}{k-1})\mathcal{M} < \mathcal{I}$
	Separate k doublings	$5k$	$2k$	k	

Table 2. Complexity comparison on direct computation : Weierstrass form

Calculation	Method	Complexity			Break-Even Point
		$\mathcal{M}$	$\mathcal{S}$	$\mathcal{I}$	
$4P$	Direct computation	9	9	1	$8.6\mathcal{M} < \mathcal{I}$
	Separate 2 doublings	4	4	2	
$8P$	Direct computation	13	13	1	$6.3\mathcal{M} < \mathcal{I}$
	Separate 3 doublings	6	6	3	
$16P$	Direct computation	17	17	1	$5.4\mathcal{M} < \mathcal{I}$
	Separate 4 doublings	8	8	4	
2^kP	Direct computation	$4k+1$	$4k+1$	1	$(3.6+\frac{5.4}{k-1})\mathcal{M} < \mathcal{I}$
	Separate k doublings	$2k$	$2k$	k	

4 Scalar Multiplication with Direct Computation

4.1 The Algorithm

Using our previous formulae for direct computation of 2^kP, we can improve elliptic scalar multiplication with the sliding signed binary window method [Go98,KT92]. For example, we apply our new formulae to the window method with windows of length 4. We represent a scalar m in $P \mapsto mP$ with a *nonadjacent form* (NAF) [1]. For example, $m = (1101110111)_2$ will be represented as $m' = (100\tilde{1}000\tilde{1}00\tilde{1})_{NAF}$, where $\tilde{1}$ denotes -1.

[1] Koyama and Tsuruoka pointed out that an NAF is not necessarily the optimal representation to use [Go98,KT92]. Although it has minimal weight, allowing a few adjacent nonzeros may increase the length of zero-runs, which, in turn, would reduces the total number of additions. Their method may be useful for our scalar multiplication with direct computations of 2^kP.

Algorithm 3 describes scalar multiplication on elliptic curves using our direct computations of $2^k P$ for the case k up to 4.

Algorithm 3: Elliptic scalar multiplication combining our direct computation of $2^k P$ with the window method, and window size $k = 4$

`INPUT:` $P \in E_m(\mathbb{F}_p)$ or $E(\mathbb{F}_p)$, $m \in \mathbb{Z}$
`OUTPUT:` $mP \in E_m(\mathbb{F}_p)$ or $E(\mathbb{F}_p)$

`Step 1.` Construct NAF representation
$m = (e_t e_{t-1} \cdots e_1 e_0)_{NAF}$, $e_i \in \{-1, 0, 1\}$
`Step 2.` Precomputation
`2.1` $P_6 \leftarrow 6P$
`2.2` For i from 7 to 10 do: $P_i \leftarrow P_{i-1} + P$
`Step 3.` $P_m \leftarrow \mathcal{O}$, $i \leftarrow t$
`Step 4.` While $i \geq 3$ do the following:
`4.1` If $e_i = 0$ then:
find the longest bitstring $e_i e_{i-1} \cdots e_l$ such that $e_i = e_{i-1} = \cdots e_l = 0$, and do the following
$P_m \leftarrow 2^{i-l+1} P_m$
$i \leftarrow l - 1$
`4.2` else ($e_i \neq 0$):
If $(e_i e_{i-1} e_{i-2} e_{i-3})_{NAF} > 0$ then:
$P_m \leftarrow 16P_m + P_{(e_i e_{i-1} e_{i-2} e_{i-3})_{NAF}}$
else:
$P_m \leftarrow 16P_m - P_{|(e_i e_{i-1} e_{i-2} e_{i-3})_{NAF}|}$
$i \leftarrow i - 4$
`Step 5.` $P_m \leftarrow (e_i \cdots e_0)_{NAF} P_m$ using the traditional double-and-add method
`Step 6.` Return P_m

In Algorithm 3, we compute $16P$ directly from P in each window rather than using 4 separate doublings. In `Step 4.1` with strings of zero-runs in the scalar m_{NAF}, we should choose computations $16P$, $8P$, $4P$ or $2P$ optimally. This can be done with rules such as: 1) If a length of zero equals to 4, we compute $16P$. 2) If a length of zero equals to 3, we compute $8P$, and so on. Note that the computation for `Step 5` is inexpensive if m is large.

Using our algorithms for scalar multiplication, many of the doublings in the traditional window method will be replaced by the direct computation of $16P$. Therefore, if one computation of $16P$ is relatively faster than four doublings, scalar multiplication with our method will be significantly improved. We will examine this improvement by real implementation in the next section.

5 Complexity Comparison on Scalar Multiplication

In this section, we discuss the computational complexity of scalar multiplication using our direct computation.

Table 3. Number of computations of 2^kP, where $1 \leq k \leq 4$, and addition in the sliding signed binary window method with window length of 4

Curves	Add	$2P$	$4P$	$8P$	$16P$
160-bit	36.82	14.93	4.99	2.58	31.75
192-bit	37.63	15.29	5.06	2.64	39.54
224-bit	37.77	15.31	5.05	2.69	47.49
256-bit	41.77	15.31	5.06	2.70	55.53
384-bit	48.76	17.13	5.10	3.59	86.26
521-bit	90.39	31.34	14.51	6.94	109.87

5.1 Number of 2^kP Computations in the Window Method

Table 3 shows the number of required computations of 2^kP and additions in the sliding signed binary window method based on Algorithm 3. The window size shown in the table is 4 as an example. The numbers were counted by our implementation such that We randomly generated 10000 exponents and counted the number of operations. The averages of the numbers are shown in Table 3. In the case of a window of length 4, direct computations of $4P$, $8P$, and $16P$ can be used.

From the table, we can see that with direct computations of up to $16P$, the computational efficiency of $16P$ significantly affects scalar multiplication.

5.2 Break-Even Point

Based on the number of computations of 2^kP in scalar multiplication, given in Table 3, we compared the computational complexity of scalar multiplication. For example, in the case of a 160-bit scalar, the complexity of scalar multiplication using direct computation with $k = 4$ can be evaluated as: $C = 36.82A+14.93D_2+4.99D_4+2.58D_8+31.75D_{16}$, where A, D_2, D_4, D_8, and D_{16} denote the complexity of the computation for point addition, doubling, $4P$, $8P$, and $16P$, respectively. The complexity of those point operations can be evaluated using the algorithms given in the previous sections. For the proposed scalar multiplication with the window method, we used Algorithm 3, which is based on the sliding window method with NAF representation for a scalar.

The complexity comparisons in the case of 160-bit are described in Tables 4 and 5. By the "Traditional method", we mean a scalar multiplication using the double-and-add method in terms of affine coordinates. Again, we assume that one squaring has complexity $\mathcal{S} = 0.8\mathcal{M}$. For larger sizes, the comparison can be obtained in the same way.

Table 4. Break-even point in scalar multiplication on a 160-bit elliptic curve with Montgomery form

	Method	Complexity	Break-Even Point
Binary	Traditional	$1360\mathcal{M} + 240\mathcal{I}$	$9.3\mathcal{M} < \mathcal{I}$
	Proposed	$1686\mathcal{M} + 205\mathcal{I}$	
NAF	Traditional	$1259\mathcal{M} + 213\mathcal{I}$	$6.6\mathcal{M} < \mathcal{I}$
	Proposed	$1551\mathcal{M} + 169\mathcal{I}$	
Window with NAF	Traditional	$1195\mathcal{M} + 197\mathcal{I}$	$6.1\mathcal{M} < \mathcal{I}$
	Proposed	$1840\mathcal{M} + 91\mathcal{I}$	

Table 5. Break-even point in scalar multiplication on a 160-bit elliptic curve with Weierstrass form

	Method	Complexity	Break-Even Point
Binary	Traditional	$800\mathcal{M} + 240\mathcal{I}$	$6.6\mathcal{M} < \mathcal{I}$
	Proposed	$1030\mathcal{M} + 205\mathcal{I}$	
NAF	Traditional	$724\mathcal{M} + 213\mathcal{I}$	$7.1\mathcal{M} < \mathcal{I}$
	Proposed	$1038\mathcal{M} + 169\mathcal{I}$	
Window with NAF	Traditional	$679\mathcal{M} + 197\mathcal{I}$	$5.6\mathcal{M} < \mathcal{I}$
	Proposed	$1269\mathcal{M} + 91\mathcal{I}$	

6 Running Time

In this section, we present the running times that we obtained with our software implementation of the proposed algorithms.

The platform consisted of a 600MHz Pentium III, which has 32-bit word, using Windows 2000, Visual C++ 6.0, and MASM 6.15. The programs were written in assembly language for multi-precision integer operations, which may be time-critical in our implementation, or in ANSI C language for other operations.

We used the following domain parameters for an elliptic curve with Montgomery form.

$$
\begin{aligned}
p &= \texttt{800000000000000000000000000000000000012b} \\
A &= \texttt{49cb474d172aadfd987191a490ae0671674fe5a9} \\
B &= \texttt{17240aee6e1c8c00a7ec1df1b8721d3f90437803} \\
G_u &= \texttt{31c0186c5389ec1c81d85f4e1449390c954f7f39} \\
G_v &= \texttt{534a718a33d4e2c2089ac68e48c8f6eb101ec46d} \\
\sharp E_m(\mathbb{F}_p) &= \texttt{800000000000000000005b4c33272e33dfe2cb9c}
\end{aligned}
$$

where $(G_u, G_v) \in E_m(\mathbb{F}_p)$, and $\sharp E_m(\mathbb{F}_p)$ denotes the number of points on E_m.

Table 6. Running time of elliptic curve and field operations in msec

Curve	Elliptic (160-bit)					Field (160-bit)		
	Add	$2P$	$4P$	$8P$	$16P$	multiply	square	inversion
Montgomery	0.11	0.12	0.19	0.25	0.29	$1.92 \cdot 10^{-3}$	$1.63 \cdot 10^{-3}$	$56.0 \cdot 10^{-3}$
Weierstrass	0.093	0.094	0.13	0.16	0.20			

Table 7. Running time of scalar multiplication of a randomly selected point in msec

Curve	Binary		NAF		Window with NAF	
(160-bit)	Traditional	Proposed	Traditional	Proposed	Traditional	Proposed
Montgomery	27.4	25.7	25.0	22.7	23.3	16.7
Weierstrass	22.5	20.2	20.0	17.1	17.9	12.3

Table 8. Improvement of the performance of scalar multiplications in %

Curve (160-bit)	Binary	NAF	Window with NAF
Montgomery	6	9	28
Weierstrass	10	14	31

Table 6 shows the running times of elliptic curve and definition field operations. [2] Table 7 shows the running times of scalar multiplications.

We achieved running time reduction as shown in Table 8. As a result of our implementation with respect to Montgomery and Weierstrass form in terms of affine coordinates, we achieved running time reduced by 28% and 31%, respectively, in the scalar multiplication of the elliptic curve of size 160-bit. The proposed algorithms improved the performance of a scalar multiplication with the binary method, as well as the window method. Therefore they are effective in an restricted environment where resources are limited, such as with a smart card.

References

[AMV93] G. B. Agnew, R. C. Mullin, S. A. Vanstone, "An Implementation of Elliptic Curve Cryptosystems Over F_2^{155}", *IEEE journal on selected areas in communications*, **11**, No.5 (1993)

[CMO98] H. Cohen, A. Miyaji, T. Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates", *Advances in Cryptology – ASIACRYPT'98*, LNCS, **1514** (1998), Springer-Verlag, 51–65.

[Go98] D. M. Gordon, "A survey of fast exponentiation methods", *Journal of Algorithms*, **27** (1998), 129–146.

[2] We used classical euclidean method for field inversion. Although the prime number p has a special form which provide first implementation, we applied general method for modular reduction. Therefore further optimization will derive faster implementation.

[GP97] J. Guajardo, C. Paar, "Efficient Algorithms for Elliptic Curve Cryptosystems", *Advances in Cryptology – CRYPTO'97*, LNCS, **1294** (1997), Springer-Verlag, 342–356.

[HT99] Y. Han, P. C. Tan, "Direct Computation for Elliptic Curve Cryptosystems", Pre-proceedings of CHES'99, (1999), Springer- Verlag, 328–340.

[IEEE] IEEE P1363-2000, (2000), `http://grouper.ieee.org/ groups/1363/`

[ITTTK99] K. Itoh, M. Takenaka, N. Torii, S. Temma, Y. Kurihara, "Fast Implementation of Public-key Cryptography on a DSP TMS320C6201", *Cryptography Hardware and Embedded Systems*, LNCS, **1717** (1999), Springer-Verlag, 61–72.

[Iz99] T. Izu, "Elliptic Curve Exponentiation for Cryptosystem", *The 1999 Symposium on Cryptography and Information Security*, (1999), 275–280.

[Ko87] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48** (1987), 203–209.

[KT92] K. Koyama, Y. Tsuruoka, "Speeding up Elliptic Cryptosystems by Using a Signed Binary Window Method", *Advances in Cryptology – CRYPTO'92*, LNCS, **740** (1993), Springer-Verlag, 345–357.

[LD99] J. Lopez, R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation", *CHES'99*, LNCS, **1717** (1999), Springer-Verlag, 316–327.

[Le87] H. W. Lenstra, Jr., "Factoring integers with elliptic curves", *Ann. of Math*, **126** (1987), 649–673.

[Mi85] V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology – CRYPTO'85*, LNCS, **218** (1986), Springer-Verlag, 417–426.

[MV93] A. Menezes, A. Vanstone, "Elliptic Curve Cryptosystems and Their Implementation", *J. Cryptology*, **6** (1993), Springer-Verlag, 209–224.

[Mo87] P. L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization", *Mathematics of Computation*, **48** (1987), 243–264.

[MOC97a] A. Miyaji, T. Ono, H. Cohen, "Efficient Elliptic Curve Exponentiation (I)", *Technical Report of IEICE*, ISEC97-16, (1997)

[MOC97b] A. Miyaji, T. Ono, H. Cohen, "Efficient Elliptic Curve Exponentiation", *Advances in Cryptology – ICICS'97*, LNCS, **1334** (1997), Springer-Verlag, 282–290.

[Mu97] V. Müller, "Efficient Algorithms for Multiplication on Elliptic Curves", Proceedings of *GI - Arbeitskonferenz Chipkarten 1998*, TU München, (1998)

[OKS00] K. Okeya, H. Kurumatani, K. Sakurai, "Elliptic Curves with the Montgomery Form and Their Cryptographic Applications", *Public Key Cryptography (PKC) 2000*, LNCS, **1751** (2000), Springer-Verlag, 238–257.

[SS00] Y. Sakai, K. Sakurai, "Efficient Scalar Multiplications on Elliptic Curves without Repeated Doublings and Their Practical Performance". *Information Security and Privacy, ACISP 2000*, LNCS, **1841** (2000), Springer-Verlag, 59–63. The final version of this paper has been published [SS01].

[SS01] Y. Sakai, K. Sakurai, "Efficient Scalar Multiplications on Elliptic Curves with Direct Computations of Several Doublings". *IEICE Trans. Fundamentals*, **E84-A** No.1 (2001), 120–129. Available at http://search.ieice.or.jp/2001/files/ e120a01.htm#e84-a,1,107

[WMPW98] E. De Win, S. Mister, B. Preneel, M. Wiener, "On the Performance of Signature Schemes Based on Elliptic Curves", *Algorithmic Number Theory III*, LNCS, **1423** (1998), Springer-Verlag, 252–266.

A Computational Complexity of Direct Computations

In this appendix, we give proofs of Theorems 1 and 2. In these proofs, we ignore the cost of field additions and a subtractions, as well as the cost of multiplications by small constants.

A.1 Proof of Theorem 1

In `Step 1` of Algorithm 1, two multiplications are performed to compute $u_1 B$ and $u_1(2A + 3u_1)$. The complexity of `Step 1` involves $2\mathcal{M}$.

In `Step 2`, the following computations are performed k times to compute D_i and E_i, and $k-1$ times to compute F_i. We first perform 2 squarings to compute E_{i-1}^2 and F_{i-1}^2. If $i > 1$, we perform one multiplication to compute $(\prod_{j=0}^{i-1} E_j)^2$. Next we perform 2 multiplications for the computation of $D_{i-1}E_{i-1}^2$ and $C(\prod_{j=0}^{i-1} E_j)^2$. Note that $(\prod_{j=0}^{i-2} E_j)^2$ should be stored in the previous loop of the iteration. This gives D_i, and so the complexity of computing D_i involves $3\mathcal{M} + 2\mathcal{S}$ if $i > 1$ and $2\mathcal{M} + 2\mathcal{S}$ if $i = 1$. Next, we perform one squaring and one multiplication to compute E_{i-1}^4 and $F_{i-1}(D_{i-1}E_{i-1}^2 - D_{i-1})$. If $i = 1$, we perform one more multiplication to compute $B_2 E_0^4$. This gives E_i, and so the complexity of computing E_i involves $\mathcal{M} + \mathcal{S}$ if $i > 1$ and $2\mathcal{M} + \mathcal{S}$ if $i = 1$. Next, if $i \neq k$, we perform 3 multiplications and one squaring to compute CD_i, $B_2(\prod_{j=0}^{i-1} E_j)^2$, $(\prod_{j=0}^{i-1} E_j)^2(2^{4i}B_2(\prod_{j=0}^{i-1} E_j)^2 + 2^{2i+1}CD_i)$ and D_i^2. This gives F_i, and so the complexity of computing F_i involves $3\mathcal{M} + \mathcal{S}$.

The total complexity of `Step 2` involves $k(4\mathcal{M} + 3\mathcal{S}) + (k-1)(3\mathcal{M} + \mathcal{S})$.

In `Step 3`, we first perform $k-1$ multiplications to compute $(\prod_{i=0}^{k-1} E_i)$ and set the result to T_1. Next, two multiplications for $(\prod_{i=0}^{k-1} E_i)^3$ and $B_2(\prod_{i=0}^{k-1} E_i)^3$ are performed. Note that $(\prod_{i=0}^{k-1} E_i)^2$ has already been computed in `Step 2`. Then, we perform on inversion to compute $(2^{3k}B_2(\prod_{j=0}^{i-1} E_j)^3)^{-1}$ and set the result to T_2. Next, we perform one multiplication to compute $E_k T_2$. Then, we obtain v_{2^k}. The complexity of computing v_{2^k} involves $(k-1)\mathcal{M} + 3\mathcal{M} + \mathcal{I}$.

To compute u_{2^k}, we perform 3 multiplications to compute $D_k B$, $D_k B T_1$, and $D_k B T_1 T_2$. Then, we obtain u_{2^k}. The complexity of computing u_{2^k} involves $3\mathcal{M}$.

According to above computation, the complexity of Algorithm 1 involves $(8k + 4)\mathcal{M} + (4k-1)\mathcal{S} + \mathcal{I}$.

A.2 Proof of Theorem 2

In `Step 1` of Algorithm 2, one squaring is performed to compute x_1^2. The complexity of `Step 1` involves $\mathcal{S}$.

In `Step 2`, the following computations are performed k times to compute A_i and B_i, and $k-1$ times to compute C_i. First, we perform 3 squarings to compute B_{i-1}^2, B_{i-1}^4, and C_{i-1}^2. Second, we perform one multiplication to compute $A_{i-1}B_{i-1}^2$. Then we obtain A_i. Third, we perform one multiplication to compute $C_{i-1}(A_i - 4A_{i-1}B_{i-1}^2)$. Then we obtain B_i. Next, we perform one squaring to compute A_i^2. If $i = 1$, we perform one multiplication to compute aB_1^4 and set the result to U, and if $i > 1$, we perform one multiplication to compute UB_{i-1}^4 and set the result to U. Then, U equals $a(\prod_{j=1}^{i-1} B_j)^4$. Then we obtain C_i. The complexity of `Step 2` involves $(2\mathcal{M} + 3\mathcal{S})k + (\mathcal{M} + \mathcal{S})(k-1)$.

In `Step 3`, we first compute $\prod_{i=1}^{k} B_i$ which takes $k-1$ multiplications. Second, we perform one inversion to compute $(2^k \prod_{i=1}^{k} B_i)^{-1}$ and set the result to T. Next, we

perform one squaring to compute T^2. Next, we perform one multiplication to compute A_kT^2. Then, we obtain x_{2^k}. Finally, we perform 2 multiplications to compute B_kT^2T. Then, we obtain y_{2^k}. The complexity of `Step 3` involves $(k-1)\mathcal{M}+3\mathcal{M}+\mathcal{S}+\mathcal{I}$.

According to above computation, the complexity of Algorithm 2 involves $(4k+1)\mathcal{M}+(4k+1)\mathcal{S}+\mathcal{I}$.